WSGR Docket No. 19329-713

PATENT APPLICATION

METHOD AND APPARATUS FOR INDUCTION PROOF

Inventor(s): Bow-Yaw Wang, Citizen of Taiwan, ROC
4F No. 58, Sec 2 Academia Rd.
Nankang, Taipei, 115

Assignee: Verplex Systems, Inc.
300 Montague Expressway
Suite 100
Milpitas, CA 95035

WSGR

Wilson Sonsini Goodrich & Rosati
650 Page Mill Road
Palo Alto, CA 94304
(650) 493-9300
(650) 493-6811

Certificate of Mailing Under 37 C.F.R. §1.10

Express Mail label number EV 333 492 005 US Date of Deposit July 23, 2003 I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR § 1.10 on the date indicated above and is addressed to: Mail Stop Patent Application, Commissioner of Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Signed: Annette Palladino Date: July 23, 2003
Annette Palladino

METHOD AND APPARATUS FOR INDUCTION PROOF

BACKGROUND OF THE INVENTION

[0001] Verification is typically the most time-consuming component in a circuit design process. Failing to detect functional design errors early in the design stages usually leads to expensive re-spin of the designs. This re-spin includes diagnosis and correction of the errors, logic and physical re-synthesis, and even re-manufacturing of the chips that can be very time-consuming, costly and delay the time-to-market of a product. If the chip designs are already used in some released products, this can even lead to product recalls that are very devastating to a company.

[0002] Property checking is an approach for verifying the functionality of a circuit design. It involves proving one or more properties specified for a circuit design. A property, which can also be called an assertion, may be logical (e.g., Boolean) and/or temporal, and describes behavior of one or more signals in the circuit design.

[0003] Formal verification of a property verifies that the property holds for all combinations of input signals and sequences over time. For example, to verify that a property holds, formal verification tools or methods attempt to check all states possible during operation of the circuit design, where the operation starts from one or more initial states of the circuit design. Successfully checking all the states ensures that the property is not violated. During the state space search, if a contradiction is found, the property is disproved and a counterexample can usually be generated to demonstrate how the violation occurs.

[0004] Formal verification is therefore very useful for uncovering corner-case bugs because it determines whether or not a property is true in the circuit design by exercising all possible behavior of the circuit design. However, due to the exceedingly large and complex circuits that are being designed today, formal verification is subject to the classical state explosion problem. For example, a typical circuit design may contain hundreds of thousands of state variables (state-holding elements, i.e. flip flops), where each state variable may have one of two values, either 0 or 1. The number of possible value combinations (or states) checked by formal verification techniques is extremely large, and some states can only be reached after a very large number of transitions. It is therefore very difficult to perform state space search exhaustively for large designs. Such complexity presents memory and time constraints that make formal verification for large, but typical, circuit designs intractable.

[0005] Bounded verification, determines whether or not a property is true in the circuit design for a specific number of transitions. In contrast to the exhaustive search associated with unbounded verification, bounded verification checks the behavior exhaustively for a limited number of transitions. Its main benefit is that the limitation on the number of transitions usually greatly reduces the complexity of the verification problem. However, bounded verification fails to check the behavior beyond the limited number of transitions.

[0006] It would be desirable to combine the benefits of limiting the number of transitions checked and yet gain some assurance on the thoroughness of property checking beyond just bounded verification.

BRIEF SUMMARY OF THE INVENTION

[0007] One embodiment includes in an inductive set of one or more states a plurality of states of a circuit design. The inductive set of one or more states includes at least states passing a first property of the circuit design. States of the inductive set passing at least the first property of the circuit design are transitioning by at least one step in a forward direction, resulting in transitioned states. It is determined if the transitioned states of the inductive set pass at least the first property of the circuit design. At least the transitioning and the determining are repeated, until at least, the determining results in the transitioned states of the inductive set passing at least the first property of the circuit design.

[0008] Another embodiment transitions by at least one step, in a backward direction, states of an inductive set of at least one or more states of a circuit design passing at least a first property of the circuit design, resulting in transitioned states. It is determined if the transitioned states of the inductive set fail at least the first property of the circuit design. The transitioning and the determining are repeated, until at least, the determining results in the transitioned states of the inductive set failing at least the first property of the circuit design.

[0009] Another embodiment attempts bounded verification of one or more properties of a circuit design for at least a first number of transitions. Induction proof of the one or more properties of the circuit design for at least the first number of transitions is attempted. It is determined if the one or more properties of the circuit design are verified, based at least on the bounded verification and the induction proof.

[0010] Another embodiment includes a first iteration of transitioning by at least one step, in a backward direction, states of a first iteration of an inductive set of at least one or more states failing at least a first property of a circuit design, resulting in a first iteration of transitioned states. It is determined in a first iteration if the first iteration of transitioned states of the inductive set fail at least the first property of the circuit design.

BRIEF DESCRIPTION OF THE FIGURES

[0011] Figure 1 shows an example of verification with both bounded verification and inductive proof.

[0012] Figure 2 shows an example of forward inductive proof.

[0013] Figure 3 shows an example of backward inductive proof.

[0014] Figure 4 shows an example of a computer capable of executing inductive proof.

DETAILED DESCRIPTION OF THE INVENTION

[0015] Figure 1 shows an example of circuit verification attempting induction proof. In 110, bounded verification is attempted, of the property/properties of a circuit design, for a number of transitions. In 120, induction proof is attempted of the property/properties of the circuit design for the number of transitions. In 130, it is determined if the property/properties of the circuit design is/are verified, based on the bounded verification and the induction proof, and if the property/properties is/are not verified, a limit/limits are increased for the bounded verification and/or the induction proof, and at least part of the algorithm is repeated. The shown algorithm is illustrative, and parts can be added, removed, rearranged, and/or modified. For example, the induction proof can be attempted prior to, after, or while the bounded verification is being attempted, or some combination thereof.

[0016] Figure 2 shows an example of attempting induction proof in the forward direction. In 210, at least states passing one or more circuit design properties are included in an inductive set. In 220, states passing one or more circuit design properties are transitioned forward, by one or more transitions. This results in transitioned states. In 230, it is determined whether transitioned states pass one or more circuit design properties. In 240, the algorithm is repeated (for example, 220 and/or 230) until, at least, transitioned states pass one or more circuit design properties. The shown algorithm is illustrative, and parts can be added, removed, rearranged, and/or modified.

[0017] Many varying embodiments of figure 2 can be practiced. For example, in 230 determining whether transitioned states pass one or more circuit design properties, may not consider transitioned states resulting from transitioning of states of the inductive set failing one or more properties of the circuit design. In 230, determining whether transitioned states pass one or more circuit design properties can consider only transitioned states resulting from transitioning of states of the inductive set passing one or more properties of the circuit design.

[0018] In another embodiment, in 220, transitioning states forward may not be performed on states of the inductive set failing one or more properties of the circuit design. Also, in 220, transitioning states forward may be performed only on states of the inductive set passing one or more properties of the circuit design.

[0019] Various embodiments can differ as to what states are included in the inductive set, prior to each time 220 and/or 230 is repeated. For example, the inductive set can include transitioned states. The inductive set can include transitioned states passing one or more properties of the circuit design. The inductive set can exclude transitioned states failing one or more properties of the circuit design.

[0020] The algorithm of figure 2 can be repeated, until all transitioned states of the inductive set are determined to pass the one or more properties of the circuit design. These transitioned states of the inductive set are transitioned by a total number of transitions. In combination with this inductive proof, bounded verification is performed. In a forward direction, initial states of the circuit design are transitioned by at least the total number of transitions, resulting in a forward transitioned set of states. If the forward transitioned set of states passes the one or more properties of the circuit design, then this bounded verification, combined with the inductive proof, is sufficient to determine the circuit design to be formally verified for the one or more properties of the circuit design.

[0021] Figure 3 shows an example of attempting induction proof in the backward direction. In 310, for a first iteration, at least states failing one or more circuit design properties are transitioned backward. This results in a first iteration of transitioned states. First iteration can refer to a very first iteration, or an early iteration. In 320, it is determined for a first iteration if the first iteration of transitioned states fail one or more circuit design properties. If so, the algorithm can end 330. If not, the algorithm proceeds to 340. In 340, at least states passing one or more circuit design properties are transitioned backwards. This results in transitioned states. In 350, it is determined if transitioned states fail one or more circuit design properties. In 360, if transitioned states pass one or more circuit design properties, part of the algorithm is repeated (such as 340 and/or 350) until, at least, transitioned states fail one or more properties of the circuit design. The shown algorithm is illustrative, and parts can be added, removed, rearranged, and/or modified.

[0022] Various embodiments can differ, at some point after 310, as to what states are included in the inductive set, prior to each time 340 is repeated. For example, the inductive set can include transitioned states. The inductive set can include transitioned states passing one or more properties of the circuit design. The inductive set can exclude transitioned states failing one or more properties of the circuit design. The inductive set can exclude transitioned states able to reach, in one forward transition, any state of the circuit design failing one or more properties of the circuit design. The inductive set can include transitioned states except for transitioned states able to reach, in one forward transition, any state of the circuit design failing one or more properties of the circuit design. The inductive set can include transitioned states passing one or more properties of the circuit design except for transitioned states able to reach, in one forward transition, any state of the circuit design failing one or more properties of the circuit design. The inductive set can exclude transitioned states failing one or more properties of the circuit design and transitioned states able to reach, in one forward transition, any state of the circuit design failing one or more properties of the circuit design.

[0023] Various embodiments can differ as to what states are considered in 350 when determining if transitioned states fail one or more circuit design properties. For example, the determining process in 350 may not consider transitioned states resulting from transitioning of states of the inductive set failing one or more properties of the circuit design. The determining process in 350 may consider only transitioned states resulting from transitioning of states of the inductive set passing one or more properties of the circuit design.

[0024] Various embodiments can differ as to what states are transitioned in 340. For example, the process of transitioning may not be performed on states of the inductive set failing one or more properties of the circuit design. The process of transitioning may be performed only on states of the inductive set passing one or more properties of the circuit design.

[0025] Various embodiments can differ as to what states are included in the inductive set after the first transitioning occurs in 310 and before the subsequent transitioning of 340 occurs. The inductive set can include the first iteration of transitioned states. The inductive set can include the first iteration of transitioned states passing one or more properties of the circuit design. The inductive set can exclude the first iteration of transitioned states failing one or more properties of the circuit design.

[0026] The following discusses unbounded verification, followed by bounded verification.

[0027] Let M be a circuit design and ϕ a property. It is checked whether M satisfies ϕ. Representing the transition relation of M where S is the state space of M, uses $T_M \subseteq S \times S$. All states satisfying ϕ are denoted as $[\phi]$. Let $S_0 \subseteq S$ be the initial states of M. An exemplary BDD-based forward search algorithm is shown in example 1. Starting from the initial states S_0, the forward search algorithm first checks that all of these states satisfy ϕ. If not, false is reported. Otherwise all reached states R is updated, all states which are reachable from D within one step computed, and newly reached states considered in the next iteration. If no new state can be found, the algorithm reports true.

[0028]

```
$R = \emptyset$
$D = S_0$
while $D \neq \emptyset$
        if $D \cap [\neg\phi] \neq \emptyset$
            then
                return false
            else
                $R = R \cup D$
                $D = T_M(D)$
                $D = D \setminus R$
return true
```

[0029] Example 1: Exemplary BDD-based Forward Search Algorithm

[0030] Similarly, one can check a property by backward search. Example 2 shows an exemplary backward search algorithm. The backward search algorithm starts from all states which do not satisfy ϕ. It is first checked whether any initial states belong to $[\neg\phi]$. If so, false is reported. Otherwise the reached states R is updated, the states which can reach bad states within one step computed, and newly reached states considered. If no new state can be found, true is reported.

[0031]

```
$R = \emptyset$
$D = [\neg\phi]$
while $D \neq \emptyset$
    if $D \cap S_0 \neq \emptyset$
        then
            return false
        else
            $R = R \cup D$
            $D = T_M^{-1}(D)$
            $D = D \setminus R$
return true
```

[0032] Example 2: Exemplary BDD-based Backward Search Algorithm

[0033] The preceding examples discussing unbounded verification can be modified to perform bounded verification. The forward search algorithm can essentially compute all reachable states from S_0. If none of them violates ϕ, true is reported. The backward search algorithm can essentially compute all states which can violate ϕ. If none of the initial states can violate ϕ, true is reported. Instead of considering all reachable states or all "bad" states, bounded verification, such as a bounded model checker, only checks whether ϕ is violated within a given bounded number of transitions (also called limit or a total of steps or a number of steps). A bounded model checker alone does not prove the property. Rather, it proves the property is true (or false) within the given number of transitions. If a design passes bounded model checking alone, it can only be said that there is no violation with traces shorter than or equal to the given bounded number of transitions (also called limit or a total of steps or a number of steps).

[0034] With bounded model checking, it is known that the design does not have any counterexample with a length shorter than or equal to the bounded number of transitions (also called limit or a total of steps or a number of steps). Inductive proof can that prove any counterexample with length longer than the given bounded number of transitions (also called limit or a total of steps or a number of steps) does not exist.

[0035] One example of forward inductive proof can be the following. Consider the algorithm shown in example 3. D is an example of an inductive set of one or more states. The algorithm starts with the entire state space. The algorithm checks whether the current set of states satisfy ϕ. If so, the algorithm terminates and reports true. Otherwise, the algorithm finds the subset which satisfies ϕ, and then computes the successors of the subset.

[0036] If the algorithm in example 3 terminates within ℓ iterations, then there is no counterexample with length greater or equal to ℓ. Consider the shortest one among such traces $s_0 \rightarrow \cdots \rightarrow s_n \rightarrow s_{n+1} \rightarrow \cdots \rightarrow s_{n+\ell}$ where $s_j \in [\phi]$ for $0 \leq j < n + \ell$ but $s_{n+\ell} \in [\neg\phi]$. Let $D_0 = S$ and D_i the value of D at the end of i-th iteration. Consider the following algorithm is one example of forward inductive proof:

[0037]

```
$D = S$
while $D \subseteq [\phi]$
        $D = D \cap [\phi]$
        $D = T_M(D)$
return true
```

[0038] Example 3: A BDD-based Forward Induction Algorithm

D_0	D_1	$\cdots$	D_ℓ
s_n	s_{n+1}	$\cdots$	$s_{n+\ell}$

[0039] Since $D_0 = S$, $s_n \in D_0$. For each $0 < i \leq \ell$, $s_{n+i-1} \in D_{i-1}$ by induction. But $s_{n+1} \in T_M(s_n)$. Hence $s_{n+1} \in D_i = T_M(D_{i-1})$. We have $s_{n+1} \in D_i$ for $0 \leq i \leq \ell$. But it cannot be because $D_\ell \subseteq [\phi]$ while $s_{n+\ell} \in [\neg\phi]$. A contradiction.

[0040] The inductive step can be embedded in the loop of the forward search algorithm. First, it can be checked if there is a counterexample of length ℓ. If so, false is reported. Otherwise, it is checked if D_ℓ s a subset of $[\phi]$. If so, true is reported. Otherwise, next iteration continues.

[0041] For backward inductive proof, the task remains proving that there is no counterexample of length greater than or equal to the given bounded number of transitions (also called limit or a total of steps or a number of steps). The following algorithm is one example of backward inductive proof. D is an example of an inductive set of one or more states:

[0042]

```
*firsttime = true*
D = $[\neg\phi]$
**do**
    D = $T_M^{-1}(D)$
    **if** $D \subseteq [\neg\phi]$ **then** *return true*
    $D = D \cap [\phi]$
**while** true
```

[0043] If the previous algorithm terminates within ℓ iterations, there is no counterexample of length greater than or equal to ℓ. Suppose $s_0 \rightarrow \cdots \rightarrow s_n \rightarrow s_{n+1} \rightarrow \cdots \rightarrow s_{n+\ell}$ is the shortest counterexample where $s_j \in [\phi]$ for $0 \leq j < n+\ell$ but $s_{n+\ell} \in [\neg\phi]$. Let $D_0 = [\neg\phi]$ and D_i the value of D at the end of i-th iteration. Consider the following diagram:

$$\begin{array}{cccc} D_0 & D_1 & \cdots & D_\ell \\ s_n & s_{n+1} & \cdots & s_{n+\ell} \end{array}$$

[0044] Since $D_\ell = [\neg\phi], s_{n+\ell} \in D_\ell$. For $0 < i \leq n$, $s_{n+\ell-i-1} \in D_{\ell-i+1}$ by induction hypothesis. But $s_{n+\ell-i} \in T_M^{-1}(s_{n+\ell-i+1})$. Hence $s_{n+\ell-i} \in D_{\ell-i} = T_M^{-1}(D_{\ell-i+1})$. Therefore $s_{n+i} \in D_i$ for $0 \leq i \leq \ell$. But it cannot be, because $s_n \in [\phi]$ while $D_i \subseteq [\neg\phi]$. A contradiction.

[0045] The following is another example of backward inductive proof. D is an example of an inductive set of one or more states.

[0046]

```
firsttime = true
D = [¬φ]
do
    if firsttime
      then
          $E = D$
          $D = T_M^{-1}(E)$
          firsttime = false
      else
          $E = D \cap [\phi]$
          $\hat{E} = D \cap [\neg\phi]$
          $D = T_M^{-1}(E) \cap (S \setminus T_M^{-1}(\hat{E}))$
    if $D \subseteq [\neg\phi]$ then return true
while true
```

[0047] Example 4: A BDD-based Backward Induction Algorithm

[0048] In both proofs, it is show that the final fragment of the shortest counterexample is contained in the *D*-sequence. With other properties to prune the *D*-sequence further while maintaining the containment relation, the proof can still go through and the algorithm has better chance to terminate.

[0049] The shortest counterexample is under consideration. It means that s_i can only reach $[\phi]$ within one step for $0 \leq i \leq \ell - 1$. Example 4 shows an example of an algorithm which takes the additional property into consideration.

[0050] In example 4, the set *D* is divided into two parts: those satisfying $\phi(E)$, and those satisfying $\neg\phi(\hat{E})$. From the second iteration onward, it computes the set of states which can reach *E* in one step ($T_M^{-1}(E)$) and the set of states which cannot reach $\hat{E}$ in one step ($S \setminus T_M^{-1}(\hat{E})$). Consider any shortest trace. Except the last two states, all states in the trace must reach a state satisfying ϕ but cannot reach any state violating ϕ within one step. The previous proof is still valid in the current setting.

[0051] Figure 4 shows an example of a computer 400 that can execute a inductive proof, which can be code 420. The computer 400 can be connected to a network 410. The computer 400 can execute code 420 with instructions to execute the proof flow. The computer 400 can have the code 420 preinstalled. The computer 400 can receive the code 420 over the network 410, which can be connected to the computer via a link 430, which can be a wireless and/or wired link. The code 400 can be in a temporary state (e.g., electrical, magnetic, and/or optical signal) and/or at least partly hardware, such as in a relatively permanent state (e.g., optical disk, magnetic disk, hard disk, temporary memory such as RAM, flash memory, processor). The computer 400 can have the code 420 installed via such a temporary and/or relatively permanent state hardware. Multiprocessor, multicomputer, and/or multithread implementations can be practiced.